

INSAttram

Alix ANNERAUD - Myriem ABID - Hugo LASCOUTS

23 décembre 2022

Table des matières

1	Introduction.	3
1.1	Présentation	3
1.2	Cahier des charges	3
2	Structure	4
2.1	Nommage	4
2.2	Structure générale	5
3	Organisation	7
3.1	Outils utilisés	7
3.2	Répartition du travail	7
4	Difficultés rencontrés	8
4.1	Traçage des lignes	8
4.1.1	Géométrie des lignes	8
4.1.2	Épaisseur des lignes	8
4.2	Résolution des itinéraires des passagers	9
4.3	Animation réaliste des trains	9
4.4	Optimisations	10
4.5	Interface graphiques	10
4.6	Manque de temps	10
5	Conclusion	11
6	Annexe	11
6.1	Cahier des charges détaillé	11
6.2	Analyse descendante	12

1 Introduction.

1.1 Présentation

Dans le cadre du projet informatique que nous réalisons suite aux cours de I1 et I2, nous avons décidé de développer un jeu. Le nom du projet est « INSAtram » où « Tram » fait référence au moyen de transport urbain. Le jeu consiste en la construction et gestion d'un réseau de transport ferroviaire. C'est pourquoi, l'identité graphique du jeu s'apparente aux cartes de réseau disponibles dans les transports en commun. Le concept n'est pas original puisqu'il est inspiré du jeu Mini Metro de l'éditeur Dinosaur Polo Club. L'objectif du joueur est d'acheminer des passagers vers leur destination, et ce, de manière optimale. Plusieurs stations qu'il faudra relier entre elles apparaissent au cours de la partie. Les passagers apparaissent à côté de leur station de départ et empruntent la ou les lignes qui desservent leur station afin d'atteindre leur destination.

Ce projet était à réaliser en groupe, le nôtre était composé de 3 membres. Par conséquent, en plus de la mobilisation des connaissances acquises en informatique, il était nécessaire de prendre en compte le travail de groupe et donc la communication ainsi que la répartition des tâches. Cette dimension, bien qu'indispensable, a rajouté une difficulté à la planification du projet. En effet, jusqu'ici, nous étions habitués à travailler de manière individuelle et sans trop se soucier d'établir un plan de projet de A à Z. Nous présentons dans ce rapport notre façon d'appréhender ce nouvel exercice.

1.2 Cahier des charges

Voici un rappel du cahier des charges. La version détaillée se trouve en annexe. Les parties soulignées sont les fonctionnalités optionnelles et seront réalisées si l'avancement du projet le permet.

- **Interface graphique :** L'interface graphique comportera les éléments suivants :
 - Un menu de démarrage permettant de démarrer une partie.
 - Un terrain de jeu contenant tous les éléments de la partie (score, temps, inventaire).
 - Un bilan à la fin de la partie (au game over) affichant le score du joueur.
 - La présence de couleurs tout en gardant une interface épurée.
- **Temps :** Organisé en jours et semaines (jour : ~20 secondes, 1 semaine : ~140 secondes). Il détermine en partie le score final du joueur. Il est possible de mettre en pause et de reprendre le temps.
- **Son :** Le jeu comprend une musique de fond apaisante ainsi pour mettre le joueur dans l'ambiance du jeu.
- **Objets :** Ce sont les éléments avec lesquels le joueur peut interagir. Tous les objets sont présents en début de partie en quantité limitée et peuvent être obtenus à la fin de chaque semaine. Les objets seront les suivants :
 - **Locomotives :** Elles effectuent des allers-retours sur les lignes afin d'acheminer les voyageurs de station en station.
 - **Lignes :** Ce sont les infrastructures permettant aux véhicules de circuler de station en station. Le joueur les dessine et peut les modifier au cours de la partie.
 - **Wagons :** Ce sont des véhicules pouvant être ajoutés à une locomotive afin d'augmenter le nombre de passagers qu'elle peut transporter en une seule fois.
 - **Ponts :** Ce sont les objets permettant à une ligne de traverser un fleuve.
- **Entités :** Ce sont les éléments qui seront disposés sur le terrain de jeu par le programme. Les entités sont les suivantes :
 - **Stations :** Elles sont de différentes formes géométriques et sont générées de manière aléatoire. Elles apparaissent à des instants t déterminés par une fonction.
 - **Passagers :** Ils sont également de différentes formes géométriques (correspondant à leur station d'arrivée) et ils apparaissent à côté des stations à des instants t déterminés par une fonction. Ces passagers disparaissent une fois arrivés à destination. L'algorithme de Dijkstra est utilisé afin de calculer l'itinéraire des passagers.
 - **Rivières :** Les différentes cartes comprennent une ou plusieurs rivières qui font office d'obstacle naturel. Elles doivent être contournées à l'aide d'un pont.
 - **-Animation :** Les entités ainsi que les objets sont animés de manière à attirer l'attention du joueur sur les différents éléments (déplacement des rames, signallement d'une gare encombrée) affichés.

- **Difficulté** : La difficulté du jeu réside dans le fait de devoir fluidifier au maximum le trafic et d'adapter le réseau en fonction des nouvelles stations qui apparaissent, le flux de passagers ainsi que des différents aléas.
- **Fin de partie** : La partie prend fin une fois qu'une station est engorgée (quand plus de 6 passagers s'y trouvent) pendant 20 secondes. Un chronomètre apparaît à côté de la/des stations surchargée(s) pour indiquer le temps restant au joueur. Si ce temps est dépassé alors que la station est encore encombrée, le joueur perd.

2 Structure

Au début du projet, nous avons essayé d'être le plus rigoureux possible lorsque l'on a déterminé la structure du projet. Cependant, certaines parties ne pouvaient être prédites lorsque le projet était encore au stade de simple idée. Nous donc dû apporter des modifications à notre analyse descendante initiale, la version qui a été mise à jour se trouve en annexe du rapport. Nous avons également veillé à séparer le code en plusieurs unités et à laisser des commentaires dans le but d'avoir un code lisible et compréhensible. Certains « writeln » qui ne sont pas utiles à l'exécution du code ont été ajoutés afin d'afficher du texte dans le terminal pour faciliter la compréhension du déroulement de la partie.

2.1 Nommage

Afin que le code soit structuré et facile à comprendre, nous avons adopté une convention d'écriture qui suit les règles suivantes :

- Langage : l'écriture du code en anglais nous semble assez naturelle, pour des raisons d'homogénéité avec le langage de programmation lui-même. Cependant, afin de faciliter la compréhension du code lors de la correction, nous avons utilisé du français pour les commentaires. Ces derniers suivent une certaine structure. Pour un commentaire faisant office de titre :

```
// - Section
```

```
// - - Sous-section
```

```
// Commentaire standard.
```

```
Procedure Station_Render();
```

- Constantes : Les constantes sont nommées

```
Const Station_Maximum
```

- Types : Tout type porte d'abord comme identifiant Type_ suivi de l'identifiant du type. Cet identifiant doit être le plus court et le plus explicite possible. Par exemple la structure contenant une station s'appelle Type_Station. Dans le cas d'un type pointeur, il suffit de rajouter _Pointer au nom du type de base :

```
Type Type_Station = Record
```

```
...
```

```
End;
```

```
Type Type_Station_Pointer = ^Type_Station;
```

- Fonctions et procédures : Les fonctions et procédures portent généralement le nom de l'objet sur lequel elles agissent ou de l'unité concernée, suivi d'un verbe (en général : Get, Set, Refresh, Render ...) indiquant l'action effectuée et est éventuellement suivi de précision sur l'action :

```
// Procedure rafraichissant les graphismes
```

```
Procedure Graphics_Refresh();
```

```
// Procedure qui rend graphiquement une station
```

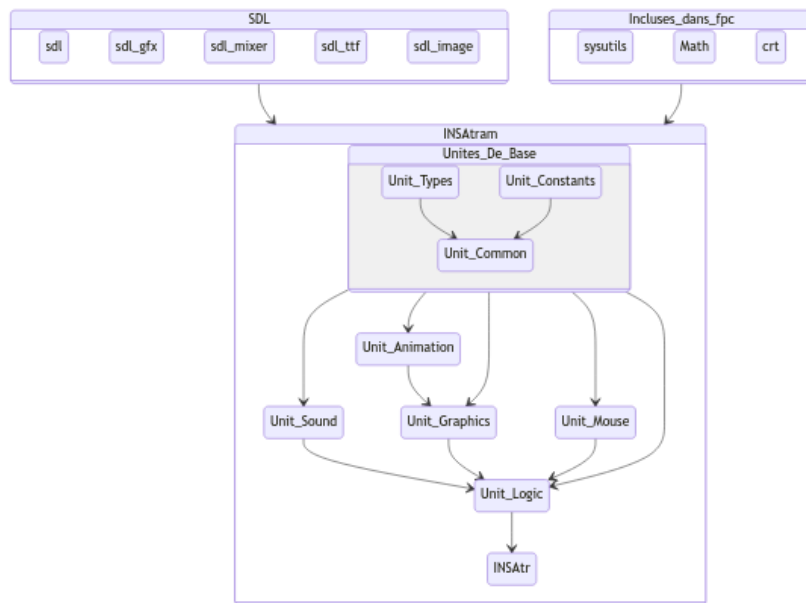
```
Procedure Station_Render();
```

2.2 Structure générale

Le projet est structuré en plusieurs fichiers de la manière suivante :

- **Unit _Types** : Unité contenant tous les types
- **Unit _Constants** : Unité contenant toutes les constantes du jeu.
- **Unit _Common** : Unité contenant toutes les fonctions « élémentaires » permettant de simplifier le développement dans les autres unités.
- **Unit _Mouse** : Unité regroupant toutes les fonctions responsables de la prise en charge de la souris.
- **Unit _Animation** : Unité regroupant toutes les fonctions responsable des animations (déplacement des trains,).
- **Unit _Graphics** : Unité regroupant toutes les fonctions relatives aux graphismes.
- **Unit _Logic** : Unité regroupant toute la logique du jeu. C'est l'unité « mère » du jeu, celle qui est directement incluse dans le programme principal : « INSAtram.pas ».

FIGURE 1 – Arbre des dépendances



3 Organisation

3.1 Outils utilisés

Ce projet étant un projet plus complexe que ce que nous avons eu à programmer jusqu'ici, nous avons fait appel à plusieurs outils afin de répondre au mieux aux besoins du projet. Voici la liste des outils utilisés pour le développement du jeu :

- Visual Studio Code : Un éditeur de code open source et gratuit, très extensible et personnalisable.
- Git : Logiciel de gestion de répertoire de code. Open source et performant.
- GitHub : Plateforme en ligne d'hébergement de répertoire de code Git.
- Live Share : Une extension de Visual Studio Code permettant une collaboration en direct (à la manière d'un Google Docs).
- Ptop : Le formateur de code fourni avec le Free Pascal Compiler.
- Inkscape : Logiciel de dessin vectoriel utilisé pour la création des ressources graphiques du jeu.
- Mermaid : Outil de création dynamique de diagrammes et de graphiques utilisé pour la création de l'analyse descendante.

3.2 Répartition du travail

Afin d'avoir une répartition du travail la plus homogène possible, nous avons adopté une méthode similaire à la méthode 'Agile'. C'est-à-dire que nous avons essayé de diviser le cahier des charges en tâches élémentaires. Ces tâches étaient disposées sur un tableau virtuel dans le but de les rendre accessibles à chaque membre du groupe. Puis, chaque personne pouvait récupérer une tâche dès que la précédente était implémentée, testée et validée par les autres. Cette méthode permettait alors à chacun d'explorer différentes parties du développement et d'avoir une vision d'ensemble quant à l'avancement du projet. Cependant, certaines tâches nécessitaient des calculs et raisonnements complexes, dans ces cas-là, nous étions au moins deux à travailler dessus. De ce fait, il serait difficile d'établir une liste exhaustive des tâches effectuées par chaque membre du groupe.

Afin de ne pas avoir à être systématiquement en présentiel pour pouvoir travailler ensemble sur le projet, nous avons utilisé des outils tels que Git et GitHub pour la gestion du répertoire de code, et LiveShare afin de collaborer le plus facilement possible. C'est pour cette raison que nous avons utilisé l'éditeur de code Visual Studio Code. Bien sûr, en cas de soucis ou de problème difficile à résoudre (voir section 4), nous nous concertions et élaborions une solution ensemble.

4 Difficultés rencontrés

4.1 Traçage des lignes

4.1.1 Géométrie des lignes

Afin de donner un aspect de plan de métro aux lignes de tram, l’affichage des lignes doit se faire uniquement avec des droites horizontales, verticales et obliques (45°). Ce problème, qui nous semblait assez abstrait et arbitraire de premier abord, était finalement relativement simple à résoudre avec une disjonction des cas possibles. Voici notre démarche :

- Deux stations, représentées par les points $S_N = (S_{N,x}, S_{N,y})$ et $S_{N+1} = (S_{N+1,x}, S_{N+1,y})$ (coordonnées centrées des stations), sont séparées par les distances $\Delta_x = S_{N+1,x} - S_{N,x}$ et $\Delta_y = S_{N+1,y} - S_{N,y}$. Les indices N et $N + 1$ font référence aux positions relatives des stations dans les tableaux de pointeurs de station stockés dans les objets de type « Type_Line ». Les deux stations forment un angle $\alpha = \arctan\left(\frac{\Delta_y}{\Delta_x}\right)$ par rapport à l’axe des abscisses. Soit s_1 le segment passant par S_N et s_2 le segment passant par S_{N+1} . Dans le cas où $-135^\circ < \alpha < -45^\circ$, pour relier les deux stations avec un segment vertical/horizontal et un oblique, il existe plusieurs possibilités :
 - s_1 oblique et s_2 vertical
 - s_1 vertical et s_2 oblique
- Il est important de noter que nous avons choisi comme convention que s_1 serait toujours vertical où horizontal et s_2 oblique.
- Ainsi, le sens et la direction de s_1 et de s_2 sera alors déterminé avec α . Dans notre exemple, s_1 est horizontal et donc part à gauche de S_N . Alors que s_2
- Afin de dessiner ces segments, la SDL a besoin des coordonnées de départ et d’arrivée pour tracer un segment (donc de S_N et I pour s_1 et de I et S_{N+1} pour s_2). Notre objectif est alors de déterminer les coordonnées du point I tel que $I = (I_x, I_y)$ en fonction des coordonnées des stations S_N et S_{N+1} . Par la suite, nous appelleront I , la position intermédiaire.
- Tout d’abord, s_1 étant vertical, on a $I_x = A_x$. On place maintenant le point E , l’intersection de la droite (IB) et de la droite horizontale passant par B . On a alors $E = (A_x, B_y)$. Or, s_2 possède un angle de $\widehat{EIB} = 45^\circ$ par rapport à l’axe des ordonnées (par définition). De plus, l’angle $\widehat{IEB} = 90^\circ$ il s’agit de l’intersection droite verticale et horizontale (donc perpendiculaires). Le triangle IBE étant rectangle en E et possédant un angle à 45° , il est donc rectangle isocèle en E . Cela permet d’en déduire que $\overline{IE} = -\overline{EB}$, or $E = (A_x, B_y)$ on a donc :

$$\overline{IE} = -\overline{EB} \Rightarrow E_y - I_y = -(B_x - E_x)$$

$$\Rightarrow I_y = -B_x + E_x - E_y$$

$$\Rightarrow I_y = -B_x + A_x - B_y$$

$$\Rightarrow I_y = B_y - \Delta_x$$

On obtient $I = (A_x, B_y - \Delta_x)$.

Pour les autres configurations de α , le même raisonnement peut être appliqué. Il faut cependant faire attention aux signes des différentes composantes qui changent. Ce même raisonnement peut s’appliquer aisément à différents α .

Ainsi, pour les autres cas

- On remarque assez vite que le triangle formé en bas peut être simplifié par l’utilisation de ce qu’on a appelé un point intermédiaire. En effet, sur un plan de métro, on constate que les lignes.

4.1.2 Épaisseur des lignes

Plus tard dans le développement, nous nous sommes rendu compte que le traçage des lignes sans épaisseur (fonction `aadrawlineRGBA` de l’unité SDL GFX) rendait les lignes peu distinguables du reste. Or la SDL ne supporte pas nativement le traçage des lignes d’une épaisseur donnée. Nous nous sommes penchés sur l’unité

SDL GFX qui possède une fonction `thickLineRGBA` permettant de dessiner des lignes avec une épaisseur. Malheureusement, les liens entre l'implémentation C de la SDL et le Free Pascal Compiler étant incomplets, la fonction n'était pas accessible en Free Pascal. Ce souci n'ayant pas été anticipé, nous avons donc dû réfléchir à une solution simple pour contourner le problème. Ainsi, à partir de la fonction « `aadrawlineRGBA` » et de l'angle de la ligne, nous avons dessiné des lignes superposées les unes par rapport aux autres. Il est ainsi possible de créer une ligne épaisse. Dans le cas de lignes verticales et horizontales, il est assez aisé de dessiner une ligne épaisse en incrémentant et décrémentant x ou y autour du centre de la ligne. Cependant, dans le cas de lignes obliques, ce raisonnement n'est pas valide.

4.2 Résolution des itinéraires des passagers

Toute la difficulté du jeu reposant sur la performance du réseau créé, il était impératif pour nous que les passagers fassent preuve d'intelligence dans leur choix d'itinéraire. Dans le cas contraire, il n'y a aucune incitation à construire un plan de réseau efficace. La recherche du plus court chemin est un problème informatique classique, et de nombreux algorithmes existent pour sa résolution. Nous ne sommes pas mathématiciens ni chercheurs, alors nous avons décidé d'en adopter un assez connu : l'algorithme de Dijkstra. Cet algorithme est supposé être de complexité $O((n + a)\log(n))$. Cependant, étant donné que nous avons fait l'implémentation de cet algorithme assez théorique nous-même, en s'inspirant seulement de son fonctionnement quand il est exécuté à la main, il serait très peu surprenant d'apprendre que sa complexité s'en soit trouvée agrandie. L'écriture des fonctions et procédures liées à la résolution d'itinéraire en elle-même n'a pas présenté de difficulté majeure, l'algorithme n'étant pas excessivement compliqué à appréhender. En revanche, toutes les subtilités d'allocation mémoire, les tableaux sur plusieurs dimensions, la rigueur dans le choix des indexes, ou bien la fragmentation des différentes tâches prérequis à l'exécution de Dijkstra dans de nombreuses fonctions, eux, ont été sources de bugs ayant nécessité de longues heures de débogage. A l'heure d'écrire ce rapport, à de rares occasions, des bugs subsistent toujours et produisent parfois des itinéraires incohérents. Le taux d'occurrence de ces derniers étant faible, la jouabilité est faiblement impactée. Aucune solution n'a été trouvée pour l'heure. De la même manière, lorsqu'elle est confrontée à une correspondance, il arrive que la fonction responsable de la montée des passagers, `Passengers_Get_On`, ne fasse pas monter le passager dans le nouveau wagon.

4.3 Animation réaliste des trains

Afin que le mouvement des trains soit réaliste, au lieu d'une vitesse constante, nous avons opté pour une vitesse variable dépendant du temps. Le but étant de mimer le déplacement d'un véhicule réel. On définit alors deux constantes : t_a qui est le temps prit par le train pour accélérer, et v_m la vitesse maximale du train. Ainsi, pour le déplacement du véhicule, on peut définir trois périodes :

- L'accélération : entre 0 et t_a , la vitesse du train augmente linéairement jusqu'à atteindre : v_m .
- Croisière : entre t_a et t_d , la vitesse du train est constante et maximale : v_m .
- Décélération : entre t_d et $t_d + t_a$, la vitesse du train diminue linéairement jusqu'à atteindre : v_m .

Soit $t \in [0, t_d + t_a]$ le temps écoulé depuis le départ du train. Avec ces 3 périodes on peut définir la fonction vitesse définie par morceau :

$$v(t) = \begin{cases} v_1(t) = t \times \frac{v_m}{t_a} & \text{si } 0 \leq t \leq t_a \\ v_2(t) = v_m & \text{si } t_a < t \leq t_d \\ v_3(t) = (t - t_d) \times \frac{-v_m}{t_a} + v_m & \text{si } t_d < t \leq t_d + t_a \end{cases}, \forall t \in [0, t_d + t_a]$$

Ce qui nous intéresse, c'est d'obtenir la position du train en fonction de t . D'après nos cours de mécanique du point, la vitesse est la dérivée de la distance par le temps. On a donc :

$$v(t) = \frac{d}{dt}d(t) \Leftrightarrow d(t) = \int v(t) dt$$

$$d(t) = \begin{cases} d_1(t) = \int_0^t v_1(t) = \frac{t^2}{2} * v_m & \text{si } 0 \leq t \leq t_a \\ d_2(t) = d_1(t_a) + \int_{t_a}^t v_2(t) = d_1(t_a) + (t - t_a) * v_m & \text{si } t_a < t \leq t_d \\ d_3(t) = d_2(t_d) + \int_{t_d}^t v_3(t) = d_2(t_d) + & \text{si } t_d < t \leq t_d + t_a \end{cases}, \forall t \in [0, t_d + t_a]$$

Maintenant que nous avons la distance en fonction du temps, il suffit maintenant d'implémenter d1, d2 et d3. On utilise des instructions conditionnelles pour déterminer laquelle utiliser en fonction du temps et des domaines de définition de chacune.

4.4 Optimisations

Au fur et à mesure que le projet avançait et que les ressources graphiques se multipliaient, nous nous sommes heurtés à un problème de performance qui affectait la fluidité du jeu. Ainsi, après quelques recherches, nous avons procédé à des optimisations relativement simples :

- Tout d'abord, lorsque la SDL crée des surfaces à partir d'images importées, ces dernières ont un format de pixel (profondeur de couleur, masques, canal alpha ...) qui n'est pas forcément le même que celui de l'affichage. Or, lors du rendu (fonction « SDL_BlitSurface »), la SDL doit effectuer une conversion à la volée pour rendre la surface sur l'écran. Cette conversion nécessite beaucoup de temps de calcul. Cependant, il est possible de s'affranchir de cette conversion en effectuant la conversion dès l'import de la ressource graphique. On utilise la fonction SDL_VideoFormat pour convertir, qui prend pour paramètre la surface à convertir, et renvoie une nouvelle surface, qui est la conversion de la première. Enfin, pour que cette conversion soit réellement effective, il faut également modifier les clés ??? . Dans le cas d'utilisation de surface avec un canal Alpha (RGBA), il faut plutôt utiliser la fonction SDL_Format.
- Ensuite, il est possible d'utiliser l'accélération graphique matérielle pour certains calculs graphiques (notamment pour l'anti-crênelage, les calculs géométriques pour le dessin des formes, la copie massive de mémoire, etc.). Il faut remplacer l'option SDL_SWSURFACE par SDL_HWSURFACE dans la fonction qui instancie la fenêtre (SDL_SetVideoMode) pour activer cette fonctionnalité. Cependant, cette optimisation est conditionnée au support d'interfaces de programmation (API) graphiques spécifiques au matériel et aux pilotes d'une machine (OpenGL). Ainsi, sur des configurations plus 'exotiques' ou anciennes, il se peut que des problèmes de compatibilité empêchent le lancement du jeu.

4.5 Interface graphiques

Une interface graphique est nécessaire pour que l'utilisateur puisse interagir avec notre jeu. Or, la SDL est une librairie très rudimentaire en termes de fonctionnalités. Cependant, la création d'une interface graphique pour . Nous avons donc créé une petite abstraction pour les différents éléments de l'interface pour permettre. Cette abstraction est constituée de :

- Type :

4.6 Manque de temps

Les fonctionnalités obligatoires que nous souhaitions implémenter au début du projet ont pu être réalisées. Cependant, par faute de temps, beaucoup de fonctionnalités optionnelles n'ont pas pu être développées. Par exemple, la fonctionnalité « Chargement/Sauvegarde Partie » présente dans la première analyse descendante a été abandonnée. En effet, la gestion de la mémoire pour ce projet étant surtout dynamique, nous avons estimé que de se lancer dans l'implémentation d'un système de sauvegarde de la partie ne serait pas une idée très sage. Le type « arcade » du jeu ne s'y prête de toute manière pas vraiment. Ce manque de temps nous a également empêché de réaliser certaines optimisations dans le code, notamment pour garder en mémoire les itinéraires afin de réduire le nombre de fois où l'on calcule les itinéraires. De même, bien que commencée, la fonctionnalité de gestion des fleuves, de leurs intersections avec les lignes ainsi que de leur contournement via les tunnels comme présentée dans la première analyse descendante n'a pas pu aboutir faute de temps.

5 Conclusion

La réalisation de ce projet nous semblait compliquée voire impossible lorsque le concept nous a été expliqué. Néanmoins, cela nous a poussé à faire preuve d'autonomie et à avoir une bonne organisation afin de respecter notre cahier des charges. C'était une introduction efficace et parfois périlleuse à la gestion de projet mais également un moyen de développer nos compétences en algorithmie et de découvrir certains concepts que nous n'avions pas eu l'occasion d'explorer en cours l'année dernière. Nous avons également rencontré plusieurs difficultés au cours de la réalisation du projet ce qui nous a poussé à faire preuve d'ingéniosité et à faire des recherches afin de trouver des solutions adaptées aux problèmes qui sont survenus.

6 Annexe

6.1 Cahier des charges détaillé

Voici un rappel du cahier des charges initial. Les parties soulignées sont les fonctionnalités optionnelles et seront réalisées si l'avancement du projet le permet.

- **Interface graphique** : L'interface graphique comportera les éléments suivants :
 - **Menu de démarrage** : Il permettra de commencer une partie, de sélectionner un niveau de difficulté, de choisir une carte, de paramétrer l'échelle de l'interface graphique, de charger une partie à partir d'un fichier et également de changer le niveau du son. Il comprendra également un onglet "à propos" comprenant des crédits ainsi que des explications sur le jeu etc.
 - **Terrain de jeu** : Une fois que le joueur a lancé une partie, un terrain de jeu occupant tout l'écran sera présenté au joueur. C'est ici que les différentes entités apparaîtront au fur et à mesure. Un inventaire (interactif) et des indicateurs (statistiques de partie) sont affichés par superposition.
 - **Echelle** : Les coordonnées ainsi que la taille des différents objets graphiques seront relatives à la taille de l'écran (la proportion de l'écran occupée par l'objet). Cela permettra de rendre automatique la mise à l'échelle des éléments graphiques pour différentes configurations d'écran.
 - **Bilan** : A la fin de la partie, un écran de fin s'affiche présentant le score du joueur (calculé sur la base du nombre de semaines où le réseau était efficace et de voyageurs transportés).
 - **Couleurs** : L'interface graphique sera le plus épuré possible avec des couleurs simples et contrastées et le minimum de couleurs possibles. L'interface graphique disposera d'un mode sombre où les gammes de couleurs claires sont remplacées par des nuances plus sombres.
- **Temps** : Le temps sera organisé en jours et semaines (jour : ~20 secondes, 1 semaine : ~140 secondes). Ce système servira à déterminer en partie le score final du joueur. Il sera possible de stopper, d'accélérer ou de reprendre le temps.
- **Bilan** : Un bilan sera aussi proposé au joueur (comprenant des graphiques, cartes de fréquentation...) pour que le joueur puisse analyser ses performances.
- **Son** : Le jeu comprendra une musique de fond apaisante ainsi que des effets sonores pour mettre le joueur dans l'ambiance du jeu.
- **Objets** : Les objets sont les éléments que le joueur pourra disposer librement sur le terrain de jeu. Tous les objets seront présents en début de partie en quantité limitée et stockés dans un inventaire. A chaque semaine passée, le joueur obtiendra des récompenses qu'il pourra choisir entre 2 ou 3 objets proposés aléatoirement. Les objets seront les suivants :
 - **Locomotives** : Les locomotives sont les véhicules permettant aux voyageurs de se déplacer entre les différentes stations. Elles seront affectées par le joueur à une ligne en particulier et effectueront des allers-retours sur cette dernière.
 - **Lignes** : Les lignes seront les infrastructures permettant aux locomotives de circuler de station en station. Elles seront dessinées par le joueur et pourront être modifiées au fur et à mesure de la partie. Les lignes seront représentées à la manière d'un plan de métro, c'est-à-dire en utilisant des lignes horizontales et verticales ainsi que des angles à 45° pour les virages.
 - **Wagons** : Les wagons sont des véhicules pouvant être ajoutés à une locomotive afin d'augmenter le nombre de passagers qu'elle peut transporter en une seule fois.
 - **Ponts** : En cas d'implémentation de topologie (ex : cours d'eau) sur la carte, ils seront utiles pour les franchissements d'obstacle.

- **Technicien** : Si implémentation d’une fonctionnalité de détérioration des lignes, le technicien permet de restaurer l’état de la ligne.
- **Surdimensionnement de la gare** : Les rames passant par une gare surdimensionnée passeront moins de temps à quai. Ces gares ont également une plus grande capacité d’accueil des passagers (peut accueillir 12 passagers au lieu de 6).
- **Entités** : Les entités sont les éléments qui seront disposés sur le terrain de jeu par le programme. Le joueur ne pourra pas interagir de manière directe avec les entités, c’est-à-dire qu’elles ne seront pas modifiables par le joueur. Les entités seront les suivantes :
 - **Stations** : Les stations sont de différentes formes géométriques et sont générées de manière aléatoire. Elles apparaîtront à des instants t déterminés par une fonction (linéaire puis éventuellement logarithmique/polynomiale). Elles seront disposées sur le terrain de jeu de manière homogène, c’est-à-dire en évitant les stations superposées ou trop proches et de même type (utilisation d’une carte de fréquentation).
 - **Passagers** : Les passagers seront également de différentes formes géométriques (correspondant à leur station d’arrivée) à côté des stations à des instants t déterminés par une fonction (linéaire puis éventuellement logarithmique/polynomiale). Ces passagers disparaissent une fois arrivés à destination. Un algorithme de résolution des graphes (pondéré en fonction de l’occupation des lignes et du temps de transit entre les stations) sera utilisé afin de calculer l’itinéraire des passagers.
 - **Fleuves** : Les différentes cartes comprendront un ou plusieurs fleuves qui feront office d’obstacle naturel et devront être contournés par le joueur en utilisant les objets qu’il a à sa disposition. - Animation : Les entités ainsi que les objets seront animés de manière à attirer l’attention du joueur sur les différents éléments (déplacement des rames, signalement d’une gare encombrée) affichés. Le niveau de détail de ces animations dépendra de l’avancement du projet.
- **Difficulté** : La difficulté du jeu se trouvera dans le fait de devoir fluidifier au maximum le trafic et d’adapter le réseau en fonction des nouvelles stations qui apparaissent, le flux de passagers ainsi que des différents aléas.
 - **Niveaux de difficulté** : Le joueur aura le choix entre 3 niveaux de difficulté (facile, moyen, difficile).
 - **Détérioration des lignes** : Les lignes auront une durée de vie limitée (en fonction du nombre de rames les parcourant) et pourront être sous la contrainte d’aléas naturels (feu, inondation etc.). Le joueur pourra alors les restaurer à l’aide de l’objet “Technicien”.
 - **Fin de partie** : La partie prend fin une fois qu’une station est engorgée (quand plus de 6 passagers s’y trouvent) pendant un temps qui dépendra de la difficulté choisie. Un chronomètre apparaîtra à côté de la/des stations surchargée(s) pour indiquer le temps restant au joueur. Si ce temps est dépassé (environ 30 secondes ; varie selon la difficulté) alors que la station est encore encombrée, le joueur perd.
- **Enregistrement et chargement de parties** : Le joueur pourra enregistrer ses parties dans des fichiers de sauvegarde ainsi que les charger. Cela contribuera également à la démonstration finale du jeu afin de charger une partie en cours et de faire la démonstration.
- **Fonctionnalités cachées** : Dans ce jeu, cela pourrait être une gare qui n’apparaît que dans certaines circonstances, une blague accessible à certains joueurs chanceux...

6.2 Analyse descendante